



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



# Bài 6: Transformers

Hà Nội, 8/2021

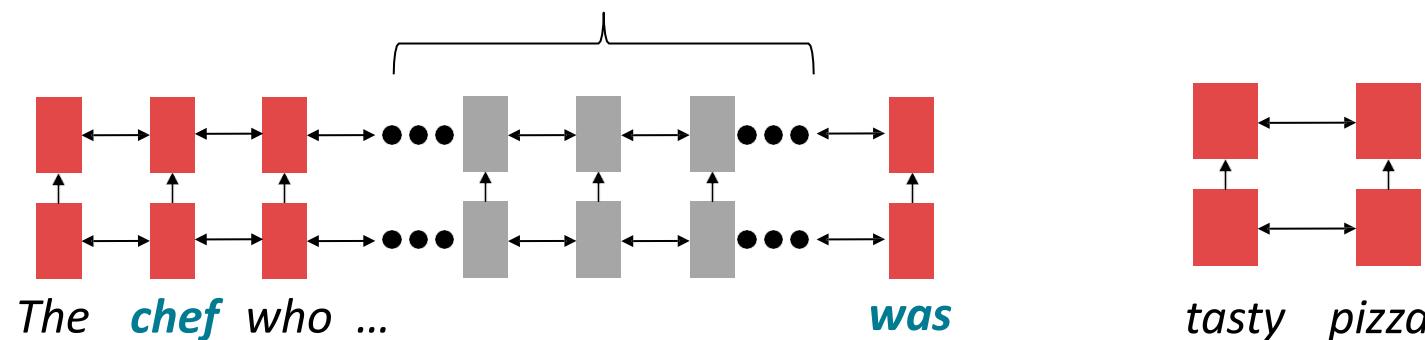
# Nội dung

1. Từ RNN tới cơ chế chú ý
2. Giới thiệu Transformers
3. Nhược điểm và một số biến thể của Transformers

# Từ RNN tới cơ chế chú ý

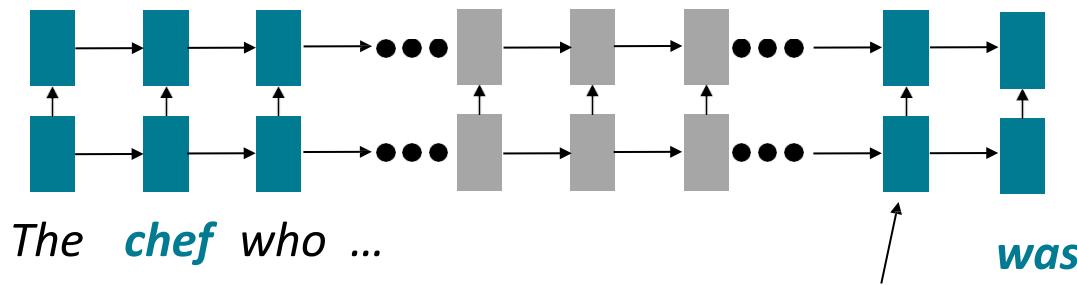
# Vấn đề của RNN: khoảng cách tương tác tuyến tính

- RNNs xử lý tuần tự từ trái sang phải
  - RNN mã hoá thông tin cục bộ một cách tuyến tính:
    - Nghĩa mỗi từ thường bị ảnh hưởng bởi các từ xung quanh
  - Vấn đề: RNNs cần  $O(\text{độ dài dãy})$  bước để các cặp từ cách xa nhau tương tác

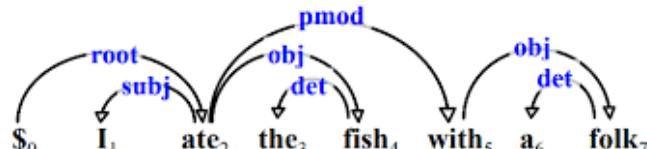


# Vấn đề của RNN: khoảng cách tương tác tuyến tính

- $O(\text{độ dài dãy})$  bước cho các cặp cách xa nhau:
  - Khó để học mối quan hệ giữa các cặp ở xa (do vấn đề triệt tiêu hoặc bùng nổ gradient!)
  - Thứ tự tuyến tính của các từ không phải là cách đúng khi phân tích câu...

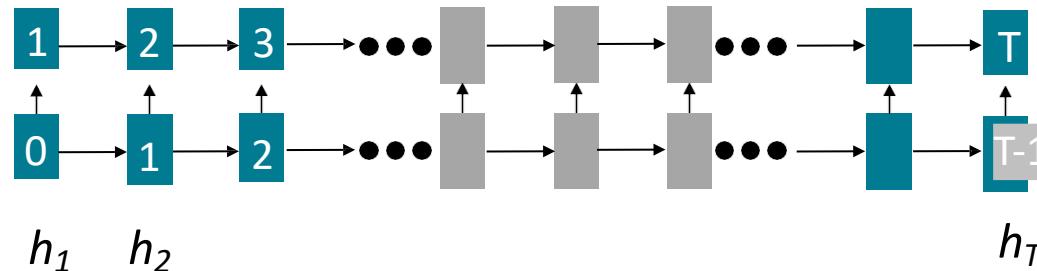


Thông tin từ **chef** biến mất qua  
nhiều lớp  $O(\text{độ dài dãy})$



# Vấn đề của RNN: Thiếu tính song song

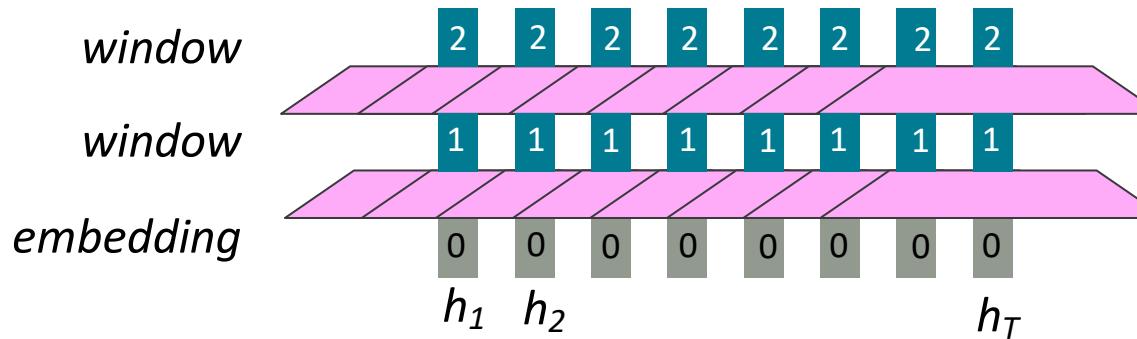
- Tính toán ngược và xuôi đều cần  $O(\text{độ dài dãy})$  phép toán không song song
  - GPUs có thể xử lý hàng loạt phép tính độc lập cùng một thời điểm!
  - Nhưng các trạng thái ẩn tương lai trong RNN không thể tính toán trước khi các trạng thái ẩn quá khứ được tính
  - Do vậy rất khó để huấn luyện trên các dữ liệu rất lớn!



Con số thể hiện số bước tối thiểu trước khi một trạng thái ẩn có thể được tính toán

# Nếu không hồi quy thì dùng cái gì? Dùng cửa sổ từ?

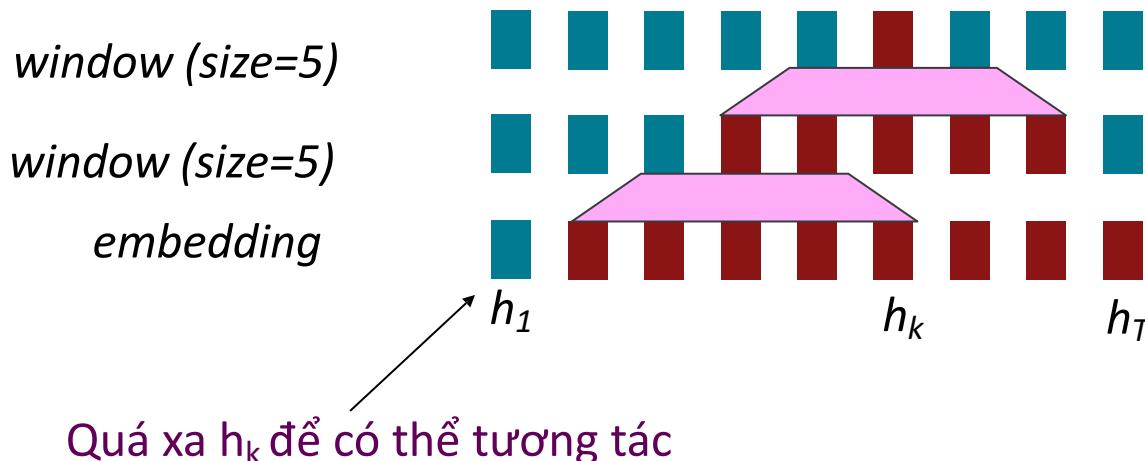
- Mô hình cửa sổ từ tổng hợp thông tin ngữ cảnh cục bộ
  - Còn được gọi là tích chập 1D
  - Số phép toán **không song song** không tăng theo độ dài dây!



Con số thể hiện số bước tối thiểu trước khi một trạng thái ẩn có thể được tính toán

# Nếu không hồi quy thì dùng cái gì? Dùng cửa sổ từ?

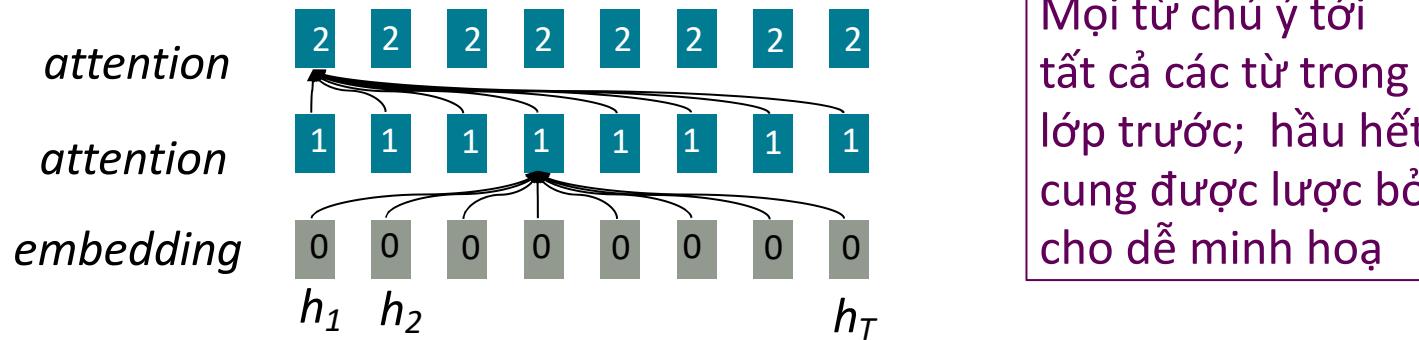
- Mô hình cửa sổ từ tổng hợp thông tin ngữ cảnh cục bộ
- Vậy những quan hệ ở khoảng cách xa thì sao?
  - Chồng nhiều cửa sổ từ cho phép tương tác với các từ ở xa hơn
- Khoảng cách tương tác xa nhất:  $O(\text{độ dài dãy} / \text{kích thước cửa sổ})$



Màu đỏ thể hiện  
những trạng thái mà  
có thể  $h_k$  tương tác

# Nếu không hồi quy thì dùng cái gì? Dùng cơ chế chú ý?

- Cơ chế chú ý xem mỗi biểu diễn của một từ như là một truy vấn (query) để truy vấn và tương tác thông tin từ một tập các giá trị.
  - Ta đã học cơ chế chú ý từ phần giải mã (decoder) sang phần mã hoá (encoder). Hôm nay ta sẽ trao đổi về cơ chế chú ý trong cùng một câu.
- Số phép toán không song song không tăng theo độ dài dãy.
- Khoảng cách tương tác xa nhất:  $O(1)$ , vì mọi từ tương tác với nhau tại mọi lớp!



# Tự chú ý (self-attention)

- Cơ chế chú ý làm việc với queries, keys và values.
  - Queries:  $q_1, q_2, \dots, q_T$  trong đó  $q_i \in \mathbb{R}^d$
  - Keys:  $k_1, k_2, \dots, k_T$  trong đó  $k_i \in \mathbb{R}^d$
  - Values:  $v_1, v_2, \dots, v_T$  trong đó  $v_i \in \mathbb{R}^d$
- Trong **tự chú ý**, queries, keys và values sinh ra từ cùng một nguồn.
- Ví dụ đầu ra của lớp trước là  $x_1, x_2, \dots, x_T$  (mỗi véc-tơ tương ứng một từ), ta có thể chọn  $q_i = k_i = v_i = x_i$ .
- Công thức tự chú ý:

$$e_{ij} = q_i^T k_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_l \exp(e_{il})}$$

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

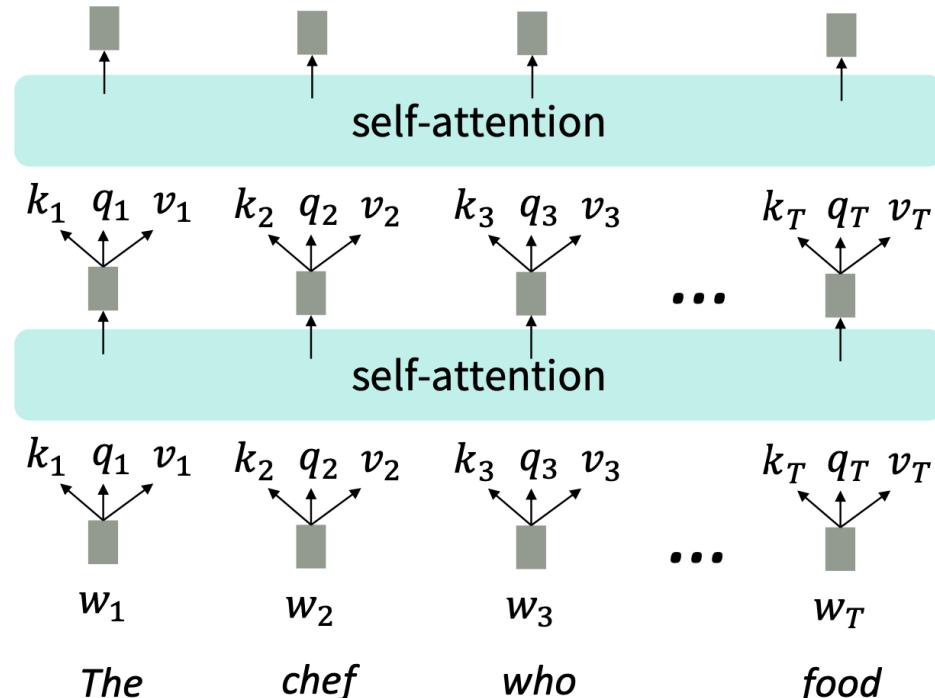
Tính tương quan  
điểm chú ý **key-query**

Tính trọng số chú ý từ  
các hệ số tương quan  
(dùng softmax)

Tính đầu ra bằng tổng có  
trọng số của các **values**

# Sử dụng Tự chú ý như một khối trong NLP

- Xếp chồng nhiều khối tự chú ý như với LSTM.
- Liệu có thể dùng cơ chế tự chú ý thay thế trực tiếp các lớp hồi quy?
  - Không, nó có một vài vấn đề mà ta sẽ xem xét.
  - Thứ nhất, tự chú ý hoạt động trên các tập hợp đầu vào, tức là nó không quan tâm tới thứ tự đầu vào.



Tự chú ý không biết thứ tự đầu vào input.

# Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

## Trở ngại

- Không có thông tin về thứ tự!



## Giải pháp

# Giải quyết vấn đề đầu tiên: thư tự dãy

- Vì tự chú ý không xây dựng thông tin về thứ tự, ta cần phải mã hoá thứ tự của câu vào trong keys, queries, và values.
- Giả sử biểu diễn mỗi vị trí trong câu bởi một véc-tơ

$p_i \in \mathbb{R}^d, i \in \{1, 2, \dots, T\}$  là các véc-tơ vị trí

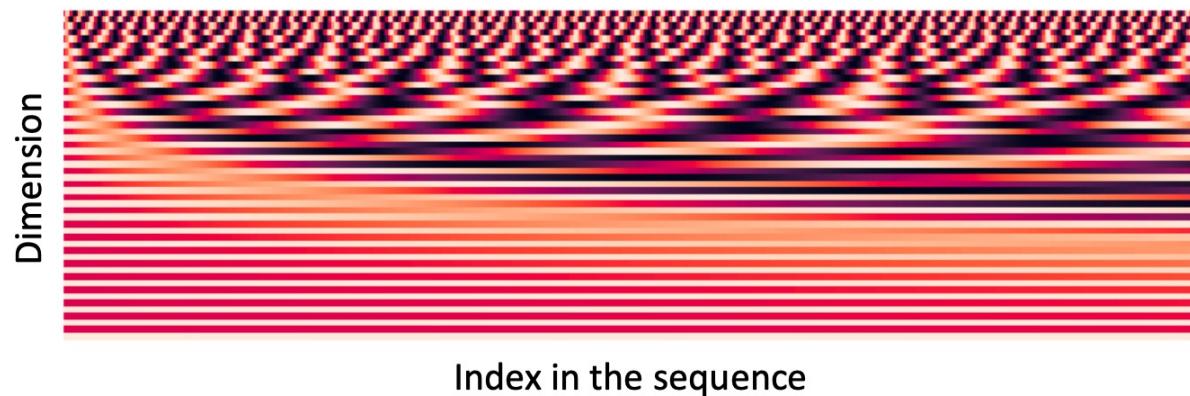
- Ta chưa cần biết  $p_i$  xác định như thế nào!
- Dễ dàng để tích hợp thông tin này vào khối tự chú ý: chỉ cần cộng  $p_i$  vào các đầu vào!
- Giả sử  $\tilde{q}_i, \tilde{k}_i, \tilde{v}_i$  là các values, keys, và queries cũ.

$$\begin{aligned}\tilde{q}_i &= q_i + p_i \\ \tilde{k}_i &= k_i + p_i \\ \tilde{v}_i &= v_i + p_i\end{aligned}$$

Trong các mạng nhiều lớp tự chú ý, chúng ta thường thêm thông tin vị trí ở lớp đầu tiên! Bạn có thể concat chúng nhưng mọi người thường dùng phép toán cộng...

# Biểu diễn thông tin vị trí thông qua hàm sin

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- **Ưu điểm:**
  - Tính chu kỳ chỉ ra rằng có thể “vị trí tuyết đối” không quá quan trọng
  - Có thể ngoại suy cho những câu dài hơn vì chu kỳ lặp đi lặp lại!
- **Nhược điểm:**
  - Không học được; việc ngoại suy cũng không thật sự hiệu quả!

# Học véc-tơ biểu diễn vị trí

- Học biểu diễn vị trí tuyệt đối: Tất cả  $p_i$  đều là véc-tơ tham số để học! Học ma trận  $p \in \mathbb{R}^{d \times T}$ , trong đó  $p_i$  là một cột của ma trận đó!
- **Ưu điểm:**
  - Linh hoạt: mỗi vị trí được học để khớp dữ liệu
- **Nhược điểm:**
  - Không thể ngoại suy cho các vị trí ngoài dải  $1, \dots, T$ .
  - Hầu hết các hệ thống dùng phương pháp này!
- Một vài công bố thử nghiệm các cách biểu diễn linh hoạt hơn:
  - Relative linear position attention [\[Shaw et al., 2018\]](#)
  - Dependency syntax-based position [\[Wang et al., 2019\]](#)

# Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

## Trở ngại

- Không có thông tin về thứ tự!
- Thiếu biến đổi phi tuyến!  
Hiện mới đơn giản là trung bình có trọng số



## Giải pháp

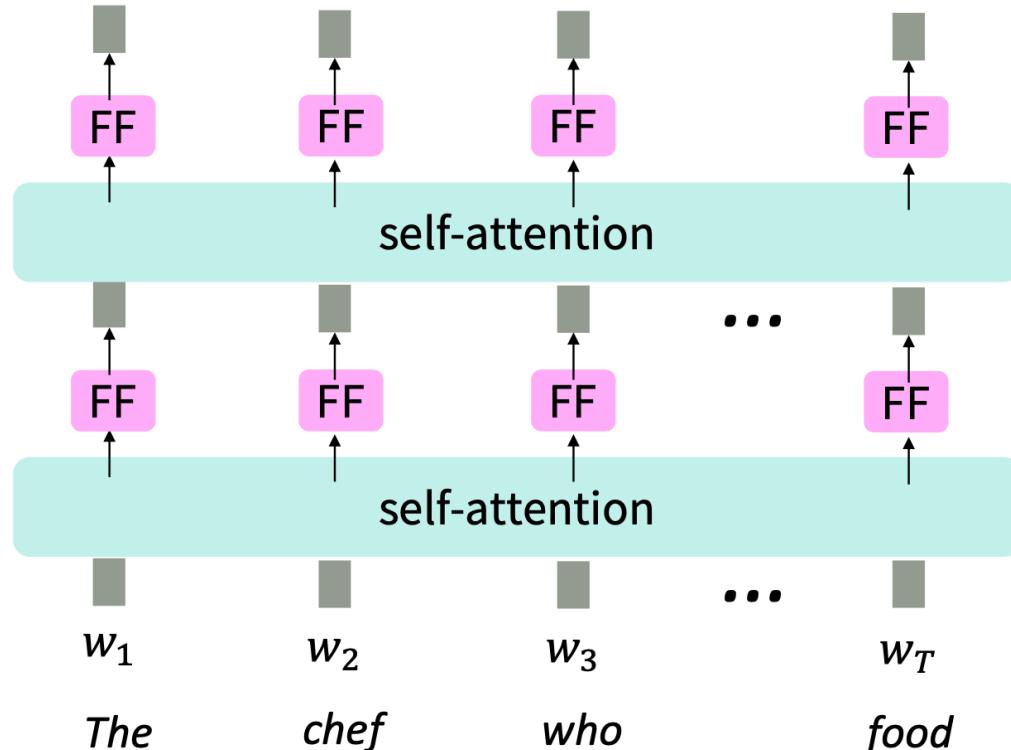
- Thêm thông tin biểu diễn vị trí vào các đầu vào



# Thêm biến đổi phi tuyến trong tự chú ý

- Lớp tự chú ý chỉ là tuyến tính. Xếp chồng bao nhiêu lớp vẫn chỉ là tuyến tính.
- Giải quyết dễ dàng: thêm mạng **feed-forward** để hậu xử lý véc-tơ đầu ra của tự chú ý.

$$\begin{aligned}m_i &= \text{MLP}(\text{output}_i) \\&= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2\end{aligned}$$



Mạng FF xử lý kết quả của lớp tự chú ý

# Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

## Trở ngại

- Không có thông tin về thứ tự!
- Thiếu biến đổi phi tuyến! Hiện mới đơn giản là trung bình có trọng số
- Cần đảm bảo “không nhìn vào tương lai” khi đoán một câu
  - Chẳng hạn trong dịch máy
  - Hoặc mô hình ngôn ngữ

## Giải pháp

- Thêm thông tin biểu diễn vị trí vào các đầu vào
- Áp dụng mạng feedforward network cho đầu ra của mỗi lớp tự chú ý.

# Đánh dấu phần tương lai

- Để dùng tự chú ý trong phần giải mã, ta phải đảm bảo không sử dụng thông tin tương lai.
- Tại mỗi bước, ta có thể đổi tập keys và queries chỉ gồm các từ quá khứ (không hiệu quả!)
- Để cho phép tính toán song song, ta có thể đánh dấu các từ tương lai bằng cách thiết lập điểm chú ý tới chúng thành  $-\infty$ .

$$e_{ij} = \begin{cases} q_i^T k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

Đối với mã hoá những từ này

[START]

Ta chỉ nhìn vào những từ này (không bị đánh dấu xám)

[START] The chef who

[The matrix of  $e_{ij}$  values]

# Đánh dấu phần tương lai

- Để dùng tự chú ý trong phần giải mã, ta phải đảm bảo không sử dụng thông tin tương lai.
- Tại mỗi bước, ta có thể đổi tập keys và queries chỉ gồm các từ quá khứ (không hiệu quả!)
- Để cho phép tính toán song song, ta có thể đánh dấu các từ tương lai bằng cách thiết lập điểm chú ý tới chúng thành  $-\infty$ .

$$e_{ij} = \begin{cases} q_i^T k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

Đối với mã hoá những từ này

Ta chỉ nhìn vào những từ này (không bị đánh dấu xám)

[START]  
The  
chef  
who

[START]	The	chef	who
The	$-\infty$	$-\infty$	$-\infty$
chef		$-\infty$	$-\infty$
who			$-\infty$

[The matrix of  $e_{ij}$  values]

# Trở ngại và giải pháp để sử dụng tự chú ý như các khối xây dựng mạng nơ-ron

## Trở ngại

- Không có thông tin về thứ tự!
- Thiếu biến đổi phi tuyến! Hiện mới đơn giản là trung bình có trọng số
- Cần đảm bảo “không nhìn vào tương lai” khi đoán một câu
  - Chẳng hạn trong dịch máy
  - Hoặc mô hình ngôn ngữ

## Giải pháp

- Thêm thông tin biểu diễn vị trí vào các đầu vào
- Áp dụng mạng feedforward network cho đầu ra của mỗi lớp tự chú ý.
- Đánh dấu phần tương lai bằng cách đặt các trọng số chú ý bằng  $-\infty$ !

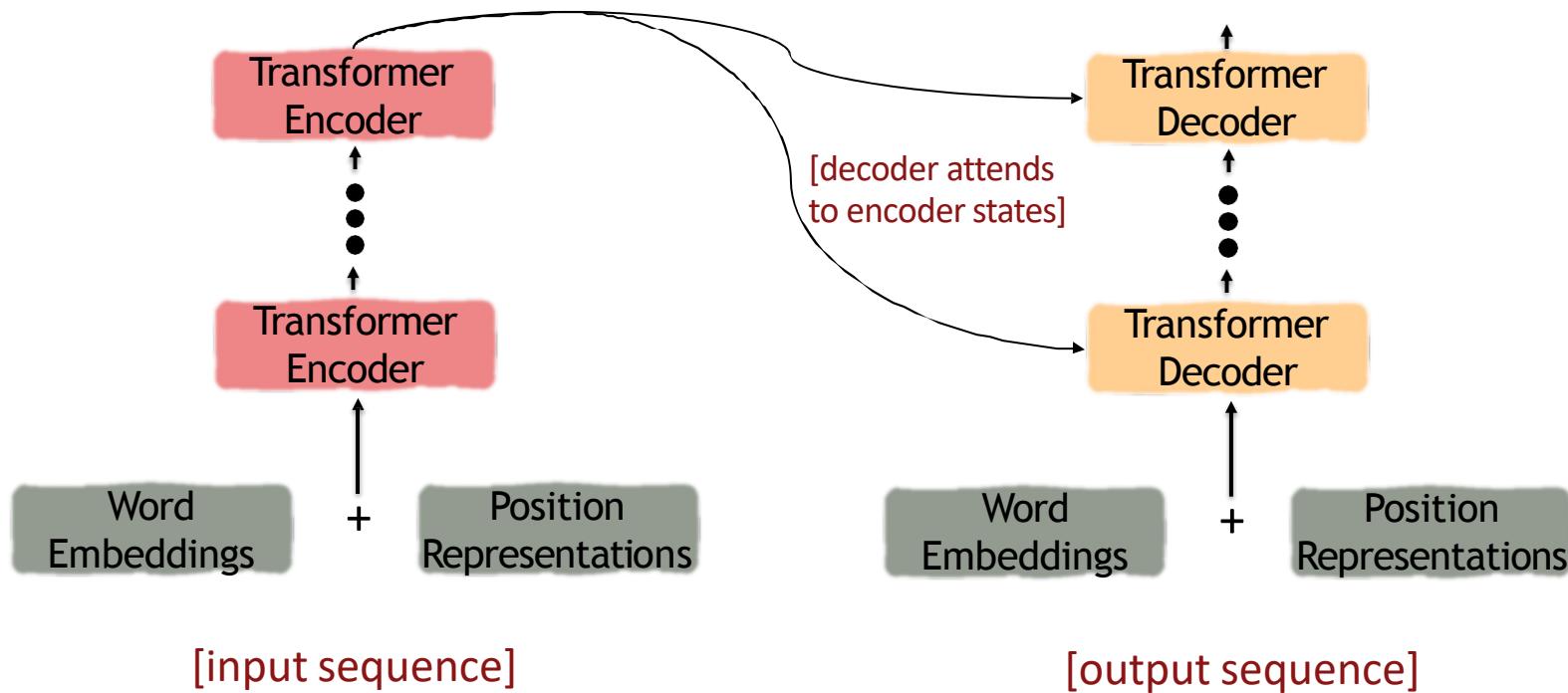
# Những thứ cần thiết cho một khối tự chú ý

- Lớp tự chú ý:
  - Thành phần cơ bản của khối.
- Biểu diễn vị trí:
  - Thể hiện thứ tự của dãy, do lớp tự chú ý không có thông tin thứ tự của đầu vào.
- Phi tuyến:
  - Áp dụng vào đầu ra của lớp tự chú ý
  - Thường được cài đặt bằng mạng feed-forward.
- Đánh dấu:
  - Để song song hoá trong khi vẫn đảm bảo không nhìn vào tương lai.
  - Giúp thông tin từ tương lai không bị “rò rỉ” về quá khứ.
- Nhưng ... đây vẫn chưa phải là mô hình Transformer mà chúng ta vẫn nghe thấy.

# Giới thiệu về Transformers

# The Transformer Encoder-Decoder [Vaswani et al., 2017]

- Trước tiên xem xét mức tổng quan



# The Transformer Encoder-Decoder [Vaswani et al., 2017]

- Còn gì trong phần giải mã (decoder) của Transformer mà ta chưa xem xét?
1. **Key-query-value attention**: làm sao để xây dựng  $k, q, v$  từ embedding của từ?
  2. Multi-headed attention: chú ý tới nhiều chỗ trong một lớp!
  3. Kỹ thuật (trick) hỗ trợ huấn luyện!
    - Kết nối tắt
    - Chuẩn hoá lớp (layer normalization)
    - Scaling the dot product
    - Các kỹ thuật này không tăng cường khả năng của mô hình mà chỉ cải thiện quá trình huấn luyện.

# Transformer Encoder: Key-Query-Value Attention

- Tự chú ý là khi keys, queries, và values được sinh ra từ cùng một nguồn. Transformer làm điều này theo một cách riêng:
  - Let  $x_1, \dots, x_T$  là véc-tơ đầu vào của Transformer encoder;  $x_i \in \mathbb{R}^d$
- Khi đó keys, queries, values xác định như sau:
  - $k_i = Kx_i$ , where  $K \in \mathbb{R}^{d \times d}$  là ma trận khoá.
  - $q_i = Qx_i$ , where  $Q \in \mathbb{R}^{d \times d}$  là ma trận truy vấn.
  - $v_i = Vx_i$ , where  $V \in \mathbb{R}^{d \times d}$  là ma trận giá trị.
- Các ma trận này cho phép biến đổi véc-tơ  $x$  theo các cách khác nhau để sử dụng chúng trong 3 vai trò khác nhau tương ứng.

# Transformer Encoder: Key-Query-Value Attention

- Tính toán key-query-value attention dưới dạng ma trận.
  - Giả sử  $X = [x_1, \dots, x_T] \in \mathbb{R}^{T \times d}$  là ghép (concat) các véc-tơ đầu vào.
  - Ký hiệu  $XK \in \mathbb{R}^{T \times d}$ ,  $XQ \in \mathbb{R}^{T \times d}$ ,  $XV \in \mathbb{R}^{T \times d}$ .
  - Đầu ra được xác định như sau: output = softmax  $(XQ (XK)^T) XV$ .

$$XQ = XQK^T X^T \in \mathbb{R}^{T \times T}$$

All pairs of  
attention scores!

$$\text{softmax} \left( \begin{matrix} XQK^T X^T \\ XV \end{matrix} \right) = \text{output} \in \mathbb{R}^{T \times d}$$

# Transformer Encoder: Multi-headed attention

- Sẽ thế nào nếu ta muốn nhìn vào nhiều chỗ trong câu cùng một lúc?
- Với từ  $i$ , lớp tự chú ý “nhìn” ở đâu  $x_i^T Q^T K x_j$  có giá trị cao, nhưng có thể chúng ta muốn chú ý vào nhiều vị trí  $j$  khác nhau cho các lý do khác nhau?
- Ta sẽ định nghĩa nhiều đầu chú ý bằng nhiều ma trận  $Q, K, V$
- Ký hiệu  $Q_l, K_l, V_l \in \mathbb{R}^{d \times (d/h)}$ , trong đó  $h$  là số đầu,  $l = 1 \dots h$ .
- Mỗi đầu chú ý thực hiện độc lập:

$$\text{output}_l = \text{softmax}(XQ_lK_l^TX^T) * XV_l, \text{ trong đó } \text{output}_l \in \mathbb{R}^{d/h}$$

- Sau đó các đầu ra được kết hợp với nhau!

$$\text{output} = Y[\text{output}_1; \dots; \text{output}_h], \text{ where } Y \in \mathbb{R}^{d \times d}$$

- Mỗi đầu “nhìn” vào những thứ khác nhau, và sinh ra các véc-tơ giá trị khác nhau.

# Transformer Encoder: Multi-headed attention

- Sẽ thế nào nếu ta muốn nhìn vào nhiều chỗ trong câu cùng một lúc?
- Với từ  $i$ , lớp tự chú ý “nhìn” ở đâu  $x_i^T Q^T K x_j$  có giá trị cao, nhưng có thể chúng ta muốn chú ý vào nhiều vị trí  $j$  khác nhau cho các lý do khác nhau?
- Ta sẽ định nghĩa nhiều đầu chú ý bằng nhiều ma trận  $Q, K, V$
- Ký hiệu  $Q_l, K_l, V_l \in \mathbb{R}^{dx(d/h)}$ , trong đó  $h$  là số đầu,  $l = 1 \dots h$ .

## Single-head attention

(just the query matrix)

$$X \quad Q = XQ$$

## Multi-head attention

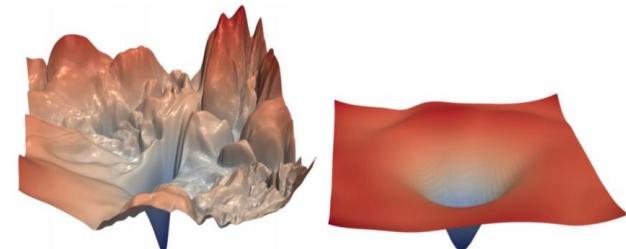
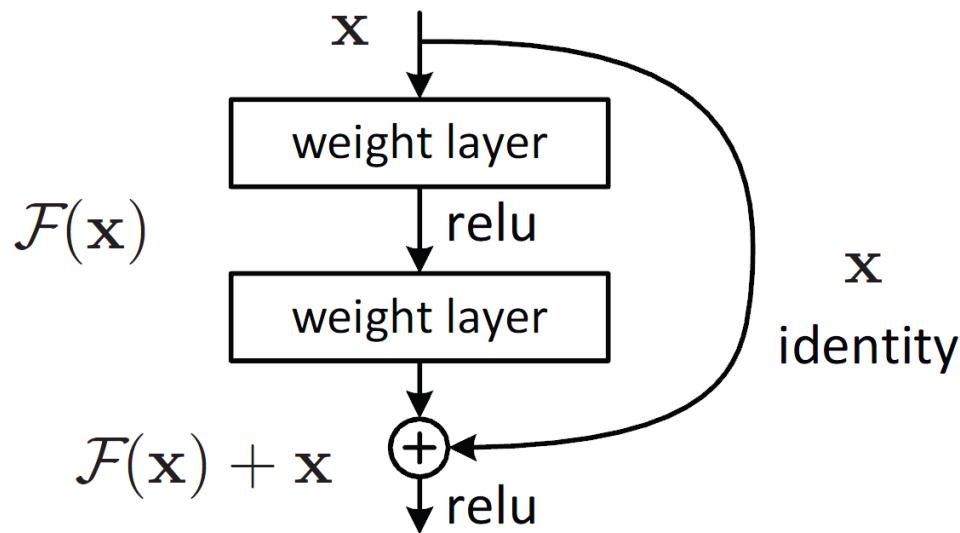
(just two heads here)

$$X \quad Q_1 \quad Q_2 = XQ_1 \quad XQ_2$$

Same amount of computation as single-head self-attention!

# Transformer Encoder: Kết nối tắt [He et al., 2016]

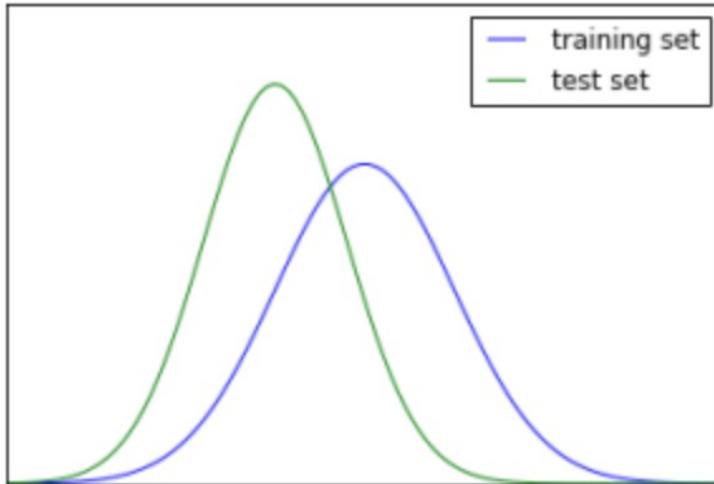
- Kết nối tắt giúp huấn luyện mô hình dễ hơn (làm bề mặt hàm mục tiêu mượt hơn)



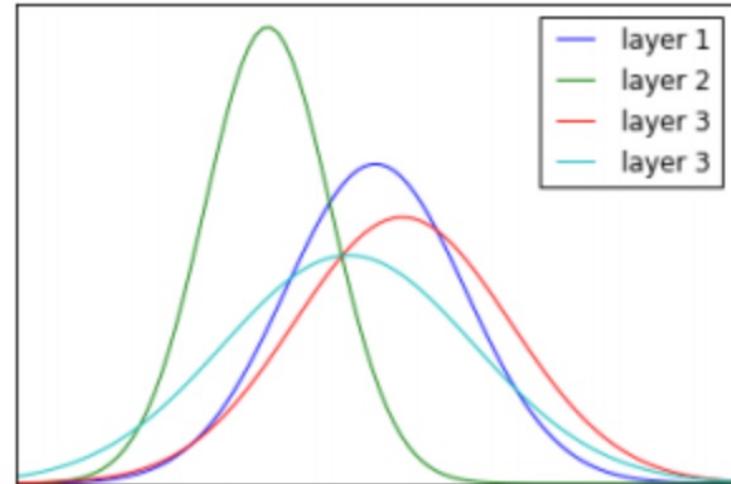
[no residuals] [residuals]  
[Loss landscape visualization,  
[Li et al., 2018](#), on a ResNet]

# Normalization

- **Covariate shift (Sự thay đổi đồng biến):** xảy ra khi một mô hình đã được đào tạo trên một tập dữ liệu có phân phối khác nhiều so với tập dữ liệu mới. Mô hình có thể phân loại sai các điểm dữ liệu trong môi trường mới.
- **Internal covariate shift:** là sự thay đổi trong phân phối đầu ra của các lớp do sự thay đổi của các tham số mạng trong quá trình huấn luyện.
- **(Batch) normalization** làm suy yếu sự thay đổi hiệp biến giữa các tham số của lớp trước và lớp sau, do đó cho phép mỗi lớp học độc lập hơn, từ đó tăng tốc quá trình học.



(a) Covariate shift



(b) Internal covariate shift

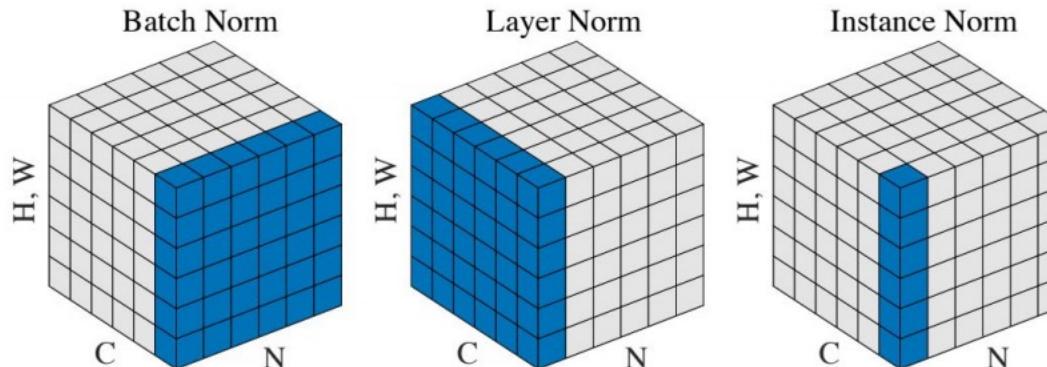
# Transformer Encoder: Chuẩn hoá lớp [Ba et al., 2016]

**Batch Normalization** for  
fully-connected networks

$$\begin{aligned} \mathbf{x}: N &\times D \\ \text{Normalize} & \downarrow \\ \boldsymbol{\mu}, \boldsymbol{\sigma}: 1 &\times D \\ \gamma, \beta: 1 &\times D \\ \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta \end{aligned}$$

**Layer Normalization** for  
fully-connected networks  
Same behavior at train and test!  
Can be used in recurrent networks

$$\begin{aligned} \mathbf{x}: N &\times D \\ \text{Normalize} & \downarrow \\ \boldsymbol{\mu}, \boldsymbol{\sigma}: N &\times 1 \\ \gamma, \beta: 1 &\times D \\ \mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta \end{aligned}$$



# Transformer Encoder: Scaled Dot Product

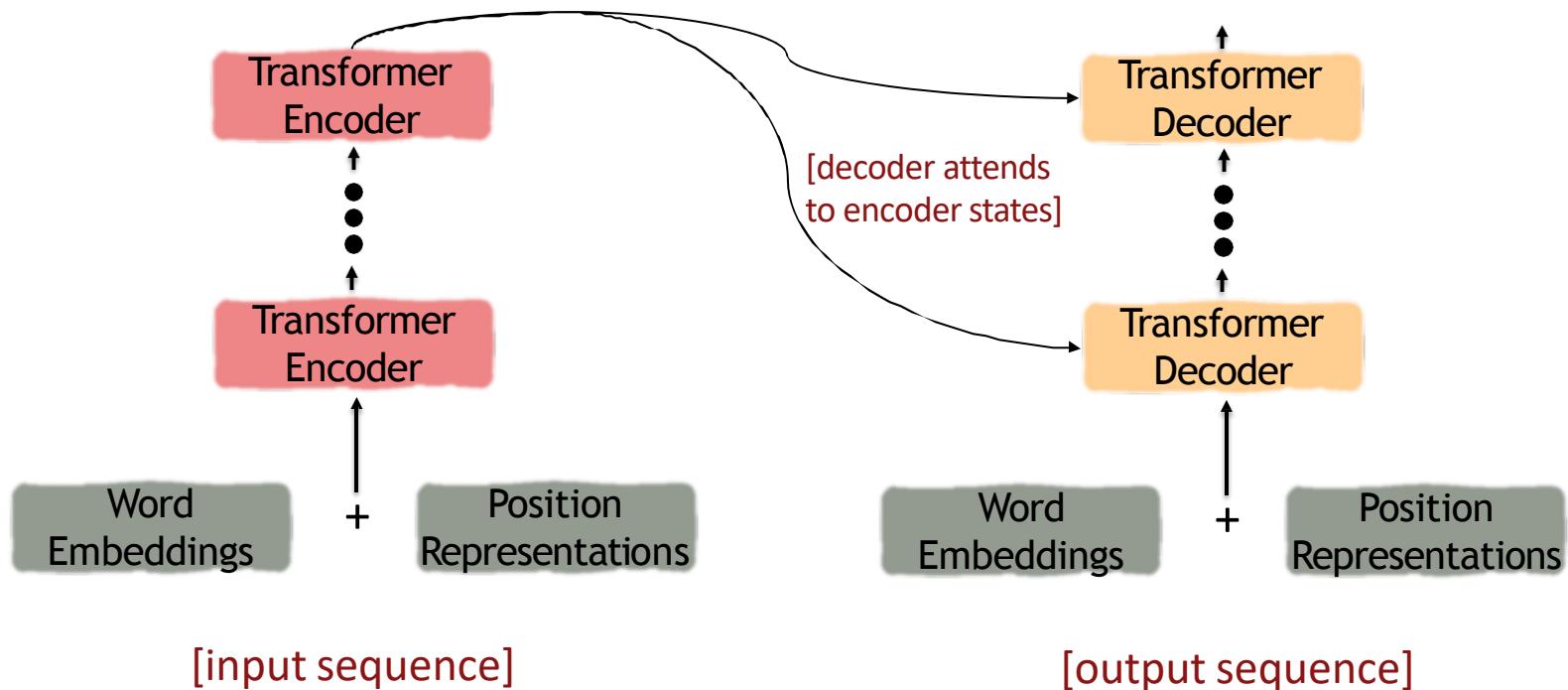
## [Vaswani et al., 2017]

- Khi số chiều  $d$  lớn, tính vô hướng giữa các véc-tơ cũng có xu hướng lớn theo.
  - Do đó đầu vào của softmax có thể rất lớn, làm hàm bão hoà và triệt tiêu gradient.
- Chia điểm chú ý cho  $d/h$ , để làm cho đầu vào của softmax bé lại

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * X V_\ell$$

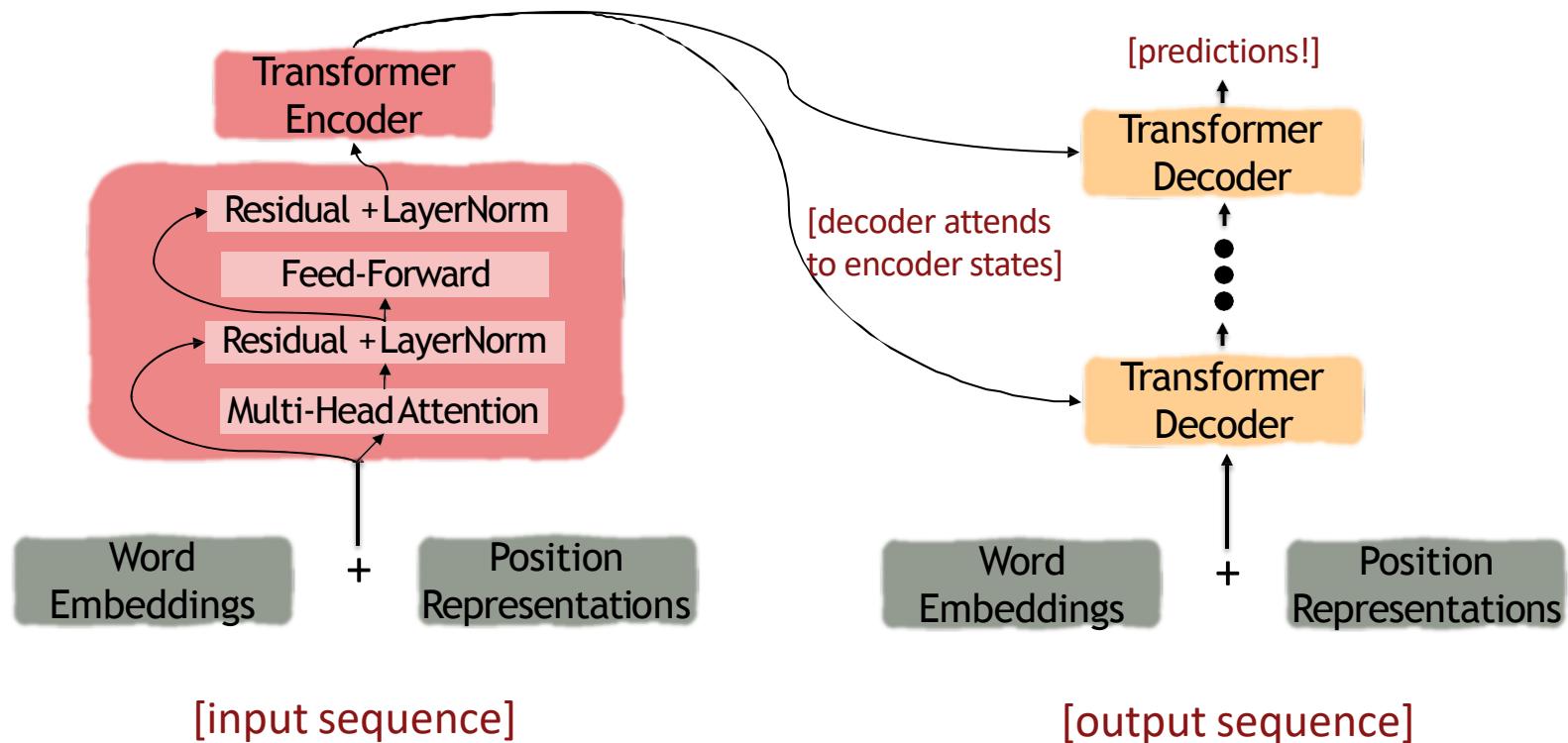
# Transformer Encoder-Decoder [Vaswani et al., 2017]

- Xem lại kỹ hơn phần mã hoá



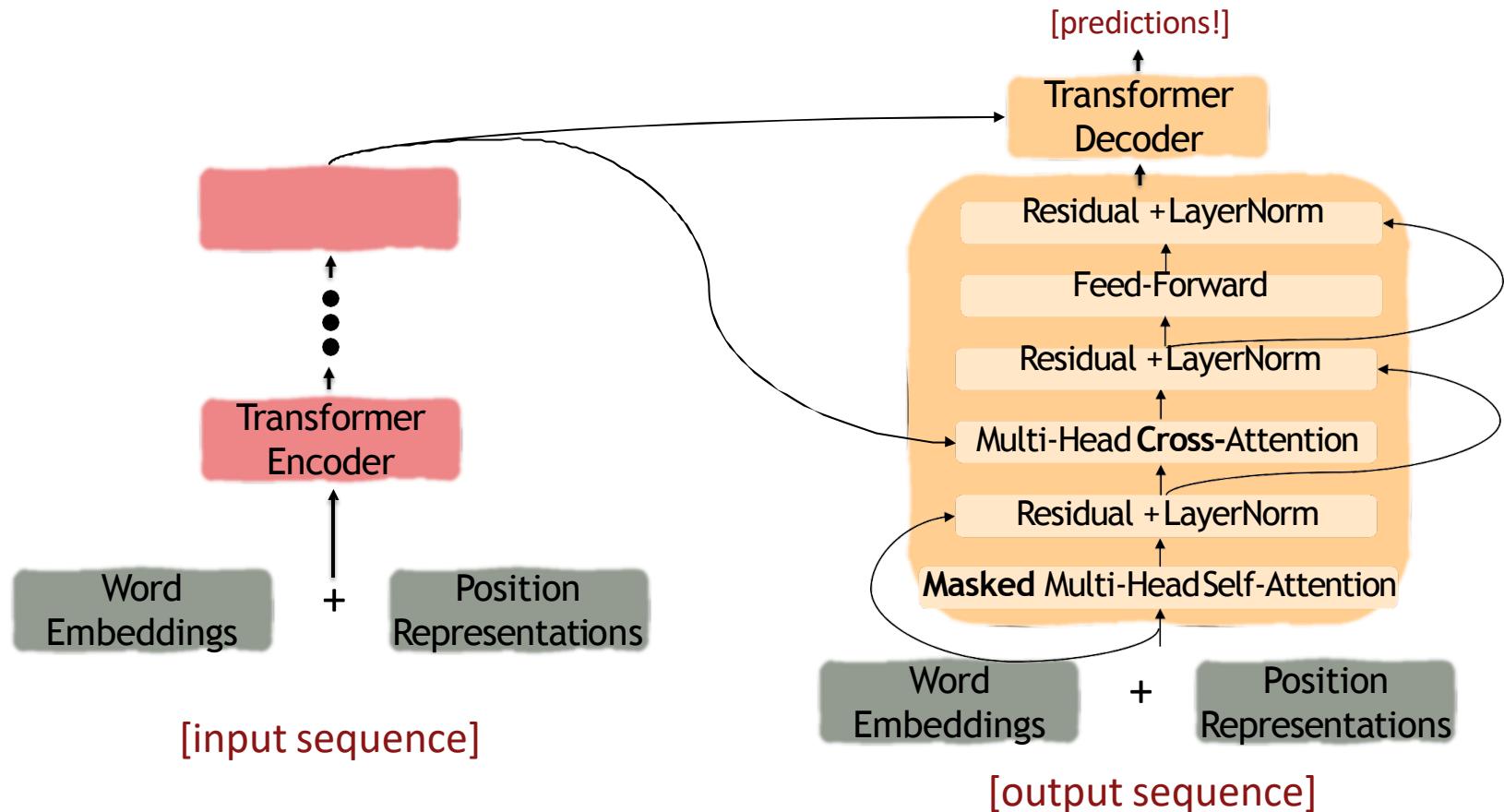
# Transformer Encoder-Decoder [Vaswani et al., 2017]

- Xem lại kỹ hơn phần mã hoá



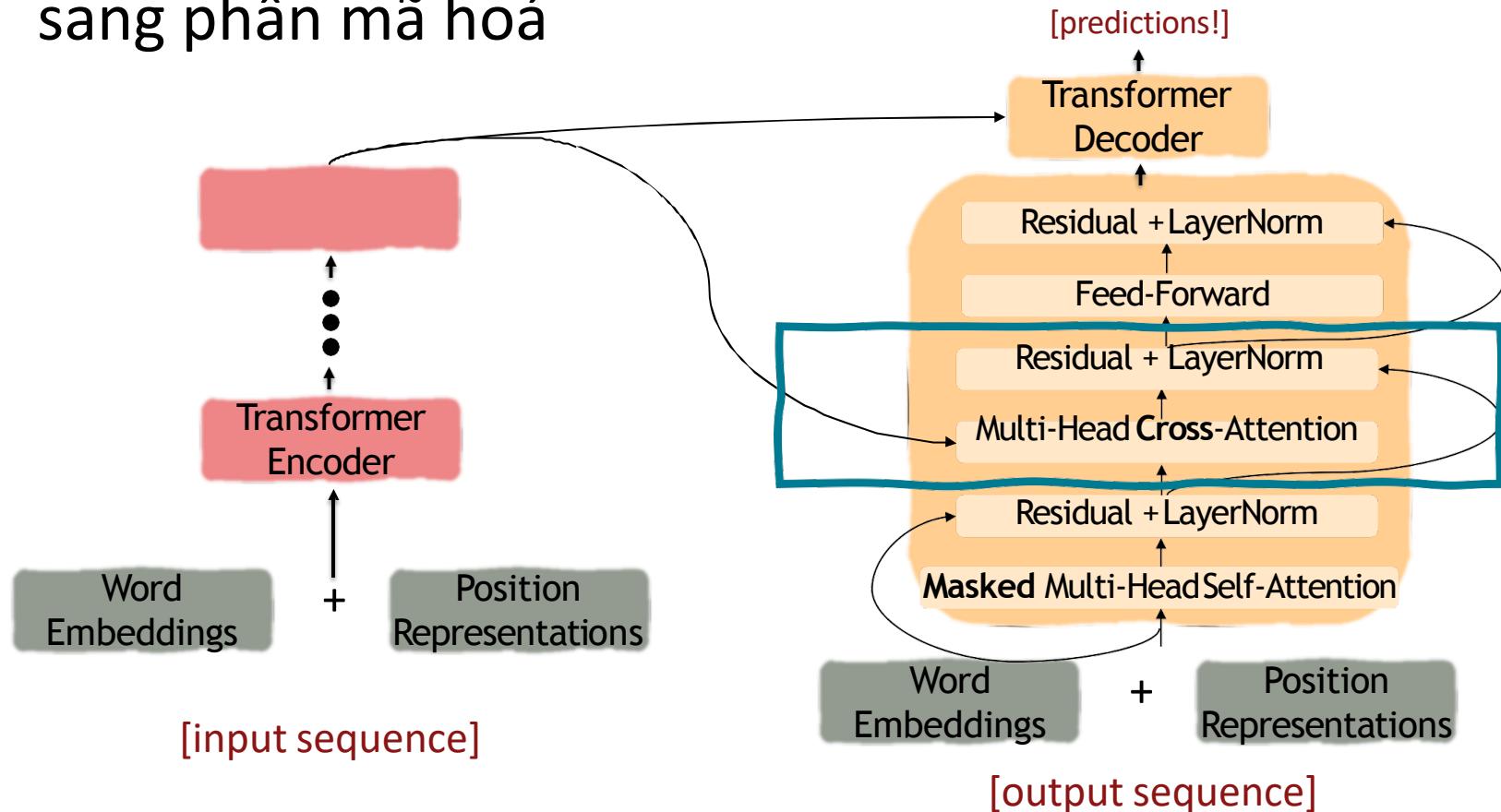
# Transformer Encoder-Decoder [Vaswani et al., 2017]

- Xem kỹ phần giải mã



# Transformer Encoder-Decoder [Vaswani et al., 2017]

- Chỉ có duy nhất phần mới là chú ý từ phần giải mã sang phần mã hóa



# Transformer Decoder: chú ý chéo (Cross-attention)

- Giả sử  $h_1, \dots, h_T \in \mathbb{R}^d$  là đầu ra phần mã hoá của Transformer;
- Giả sử  $z_1, \dots, z_T \in \mathbb{R}^d$  là đầu vào của phần giải mã Transformer
- Khi đó keys và values được tính toán từ phần mã hoá (như một bộ nhớ):
$$k_i = Kh_i, v_i = Vh_i$$
- Truy vấn được tính toán từ phần giải mã,  $q_i = Qz_i$ .

# Transformer Decoder: chú ý chéo (Cross-attention)

- Tính toán chú ý chéo dưới dạng ma trận

- Giả sử  $H = [h_1, \dots, h_T] \in \mathbb{R}^{T \times d}$  là ghép (concat) các véc-tơ mã hoá.
- Giả sử  $Z = [z_1, \dots, z_T] \in \mathbb{R}^{T \times d}$  là ghép (concat) các véc-tơ giải mã.
- Đầu ra được xác định như sau: output = softmax ( $ZQ (HK)^T HV$ ).

$$\begin{aligned} ZQ & \quad K^T H^T \\ & = ZQK^T H^T \\ & \quad \in \mathbb{R}^{T \times T} \end{aligned}$$

All pairs of attention scores!

$$\text{softmax} \left( \begin{array}{c} ZQK^T H^T \\ \vdots \end{array} \right) HV = \text{output} \in \mathbb{R}^{T \times d}$$

# Kết quả của Transformers

- Dịch máy

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

[Test sets: WMT 2014 English-German and English-French]

[Vaswani et al., 2017]

# Kết quả của Transformers

- Sinh văn bản

Model	Test perplexity	ROUGE-L
<i>seq2seq-attention, L = 500</i>	5.04952	12.7
<i>Transformer-ED, L = 500</i>	2.46645	34.2
<i>Transformer-D, L = 4000</i>	2.22216	33.6
<i>Transformer-DMCA, no MoE-layer, L = 11000</i>	2.05159	36.2
<i>Transformer-DMCA, MoE-128, L = 11000</i>	1.92871	37.9
<i>Transformer-DMCA, MoE-256, L = 7500</i>	1.90325	38.8

[Liu et al., 2018]; WikiSum dataset

# Nhược điểm và một số biến thể của Transformers

# Nhược điểm của Transformer

- **Độ phức tạp bình phương trong lớp tự chú ý:**
  - Tính toán tương tác tất cả các cặp  $O(T^2)$ !
  - Với mô hình hồi quy, độ phức tạp chỉ tuyến tính  $O(T)$ !
- Vấn đề biểu diễn vị trí:
  - liệu có thể làm tốt hơn việc biểu diễn đơn giản các vị trí tuyệt đối?
  - Relative linear position attention [Shaw et al., 2018](#)
  - Dependency syntax-based position [Wang et al., 2019](#)

# Độ phức tạp bình phương như là hàm theo độ dài dãy

- Ưu điểm của tự chú ý so với RNN là tính song song hóa cao.
- Tuy nhiên, số phép toán tăng theo  $O(T^2d)$ , trong đó  $T$  là độ dài dãy, và  $d$  số chiều embedding.

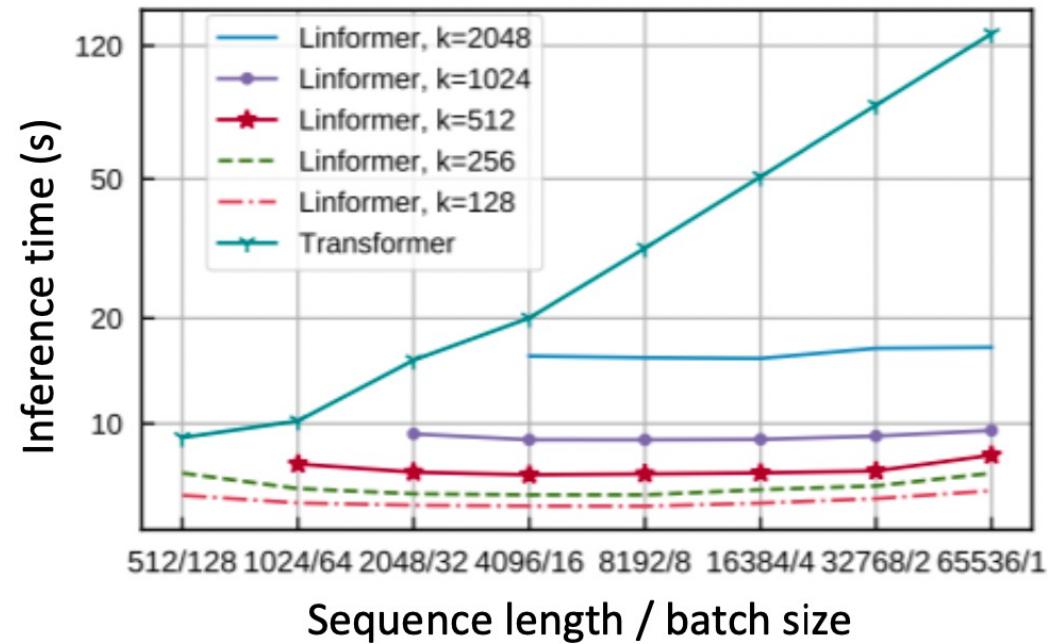
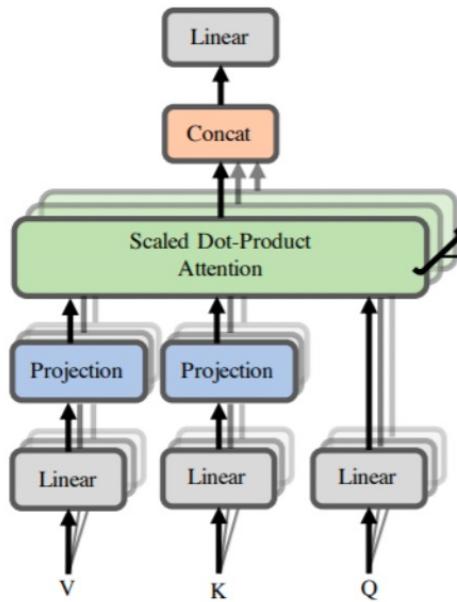
$$\begin{matrix} XQ \\ K^\top X^\top \end{matrix} = \begin{matrix} XQK^\top X^\top \\ \in \mathbb{R}^{T \times T} \end{matrix}$$

Need to compute all pairs of interactions!  
 $O(T^2d)$

- Ví dụ  $d$  khoảng 1, 000.
  - Với câu ngắn,  $T \leq 30$ ;  $T^2 \leq 900$ .
  - Thực tế, ta đặt giới hạn  $T = 512$ .
  - Nhưng nếu  $T \geq 10, 000$  thì sao? Ví dụ để xử lý các văn bản dài?

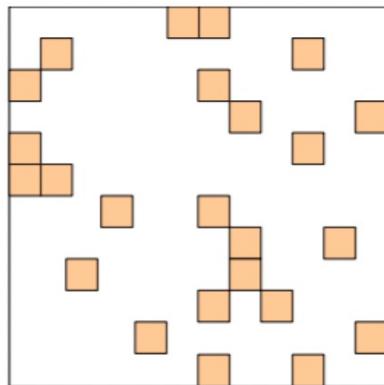
# Một số nghiên cứu gần đây giảm độ phức tạp tính toán bình phương của lớp tự chú ý

- Ví dụ, Linformer [\[Wang et al., 2020\]](#)

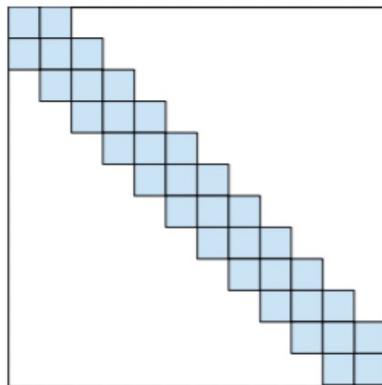


# Một số nghiên cứu gần đây giảm độ phức tạp tính toán bình phương của lớp tự chú ý

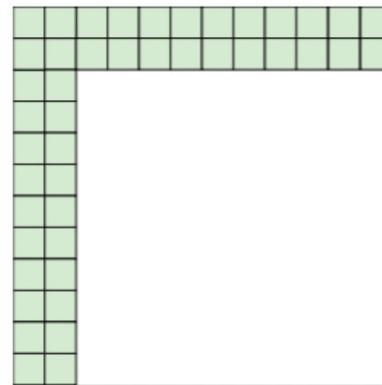
- Ví dụ, BigBird [\[Zaheer et al., 2021\]](#)



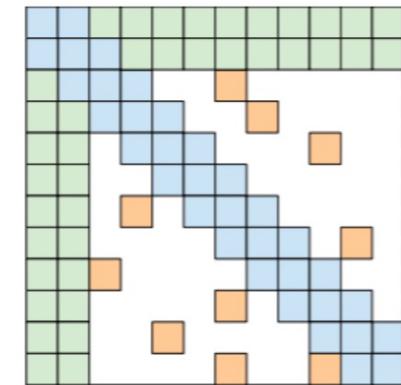
(a) Random attention



(b) Window attention



(c) Global Attention

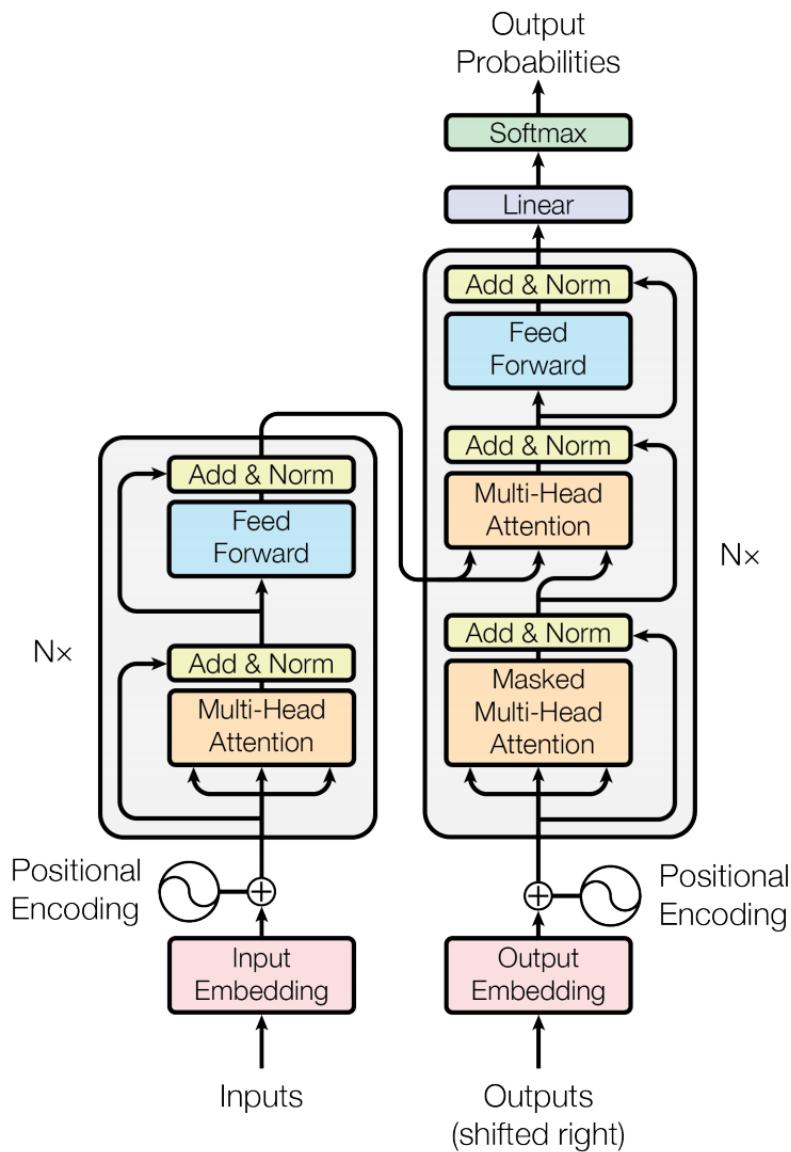


(d) BIGBIRD

# BERT (Bidirectional Encoder Representations from Transformers)

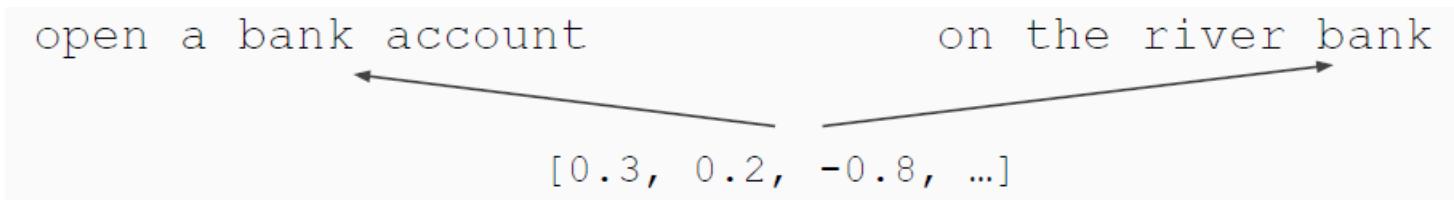
# Nhắc lại về Transformer

- Mô hình chuỗi sinh chuỗi seq2seq
- Kiến trúc Encoder-Decoder
- Tác vụ: dịch máy với dữ liệu song song
- Phân đoán các từ ở ngôn ngữ đích



# Pre-training trong NLP

- Véc tơ nhúng (Word embeddings) là thành phần cơ bản thiết yếu trong ứng dụng học sâu vào NLP
- Word embeddings (word2vec, GloVe) thường được huấn luyện trước (pre-trained) trên kho ngữ liệu khai thác các đặc trưng thống kê đồng xuất hiện
- **Vấn đề:** Word embeddings không có tính ngữ cảnh (context free manner)



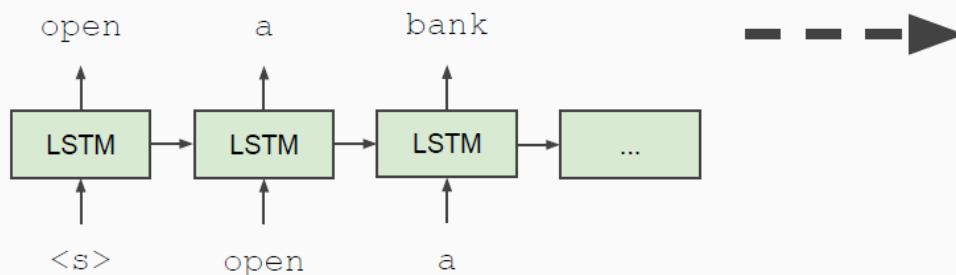
- **Mong muốn:** Huấn luyện biểu diễn có tính ngữ cảnh (*contextual*)



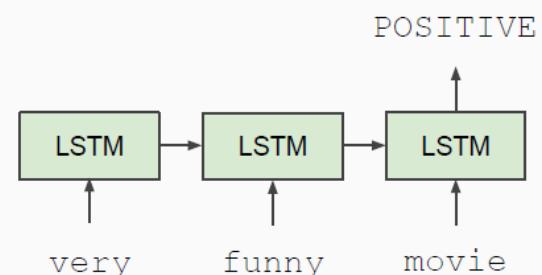
# Các nghiên cứu trước đó về biểu diễn có tính ngữ cảnh

- *Semi-Supervised Sequence Learning*, Google, 2015

## Train LSTM Language Model



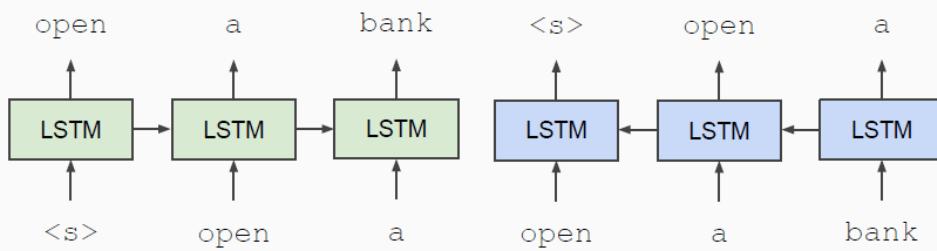
## Fine-tune on Classification Task



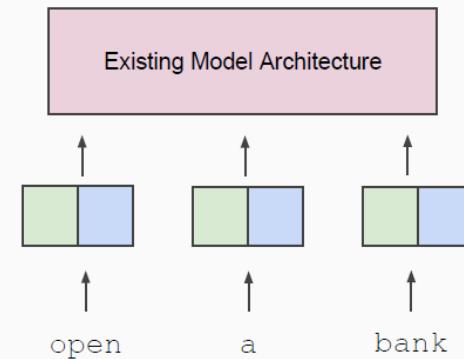
# Các nghiên cứu trước đó về biểu diễn có tính ngữ cảnh

- *ELMo: Deep Contextual Word Embeddings*, AI2 & University of Washington, 2017

**Train Separate Left-to-Right and Right-to-Left LMs**



**Apply as “Pre-trained Embeddings”**

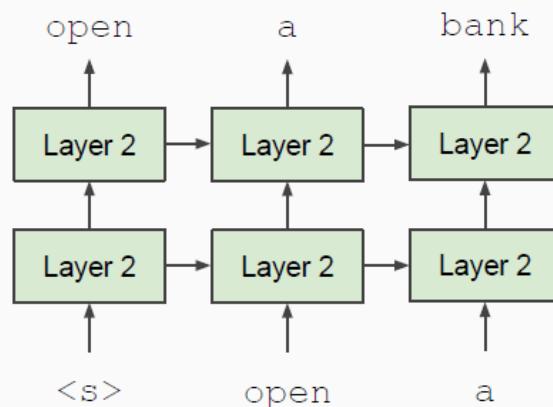


# Vấn đề với các hướng tiếp cận này

- **Vấn đề:** Các mô hình ngôn ngữ này chỉ sử dụng ngữ cảnh bên trái hoặc phải, nhưng ngôn ngữ nói chung có ngữ cảnh từ 2 hướng.

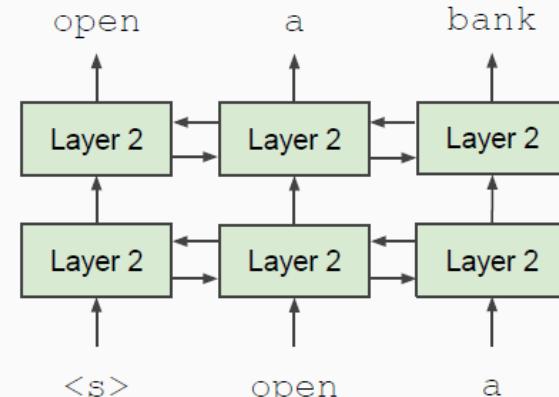
## Unidirectional context

Build representation incrementally



## Bidirectional context

Words can “see themselves”



# Masked LM

- **Giải pháp:** Che đi  $k\%$  các từ đầu vào và phán đoán các từ bị che này
    - Thường sử dụng  $k = 15\%$

the man went to the [MASK] to buy a [MASK] of milk

↑                              ↑  
store                            gallon

- Che quá ít: huấn luyện tốn kém
  - Che quá nhiều: Không đủ ngữ cảnh để học

# Masked LM

- Vấn đề: Từ bị che không xuất hiện khi fine-tuning
- Giải pháp: Che 15% nhưng không thay thế bằng token đặc biệt [MASK] 100%. Thay vào đó:
  - 80% thay thế với [MASK]
    - went to the store → went to the [MASK]
  - 10% thay thế với từ ngẫu nhiên
    - went to the store → went to the running
  - 10% giữ nguyên
    - went to the store → went to the store

# Phán đoán câu tiếp theo

- Để tìm hiểu mối quan hệ giữa các câu, hãy dự đoán xem Câu B là câu thực tế tiếp nối Câu A hay là một câu ngẫu nhiên

**Sentence A** = The man went to the store.

**Sentence B** = He bought a gallon of milk.

**Label** = IsNextSentence

**Sentence A** = The man went to the store.

**Sentence B** = Penguins are flightless.

**Label** = NotNextSentence

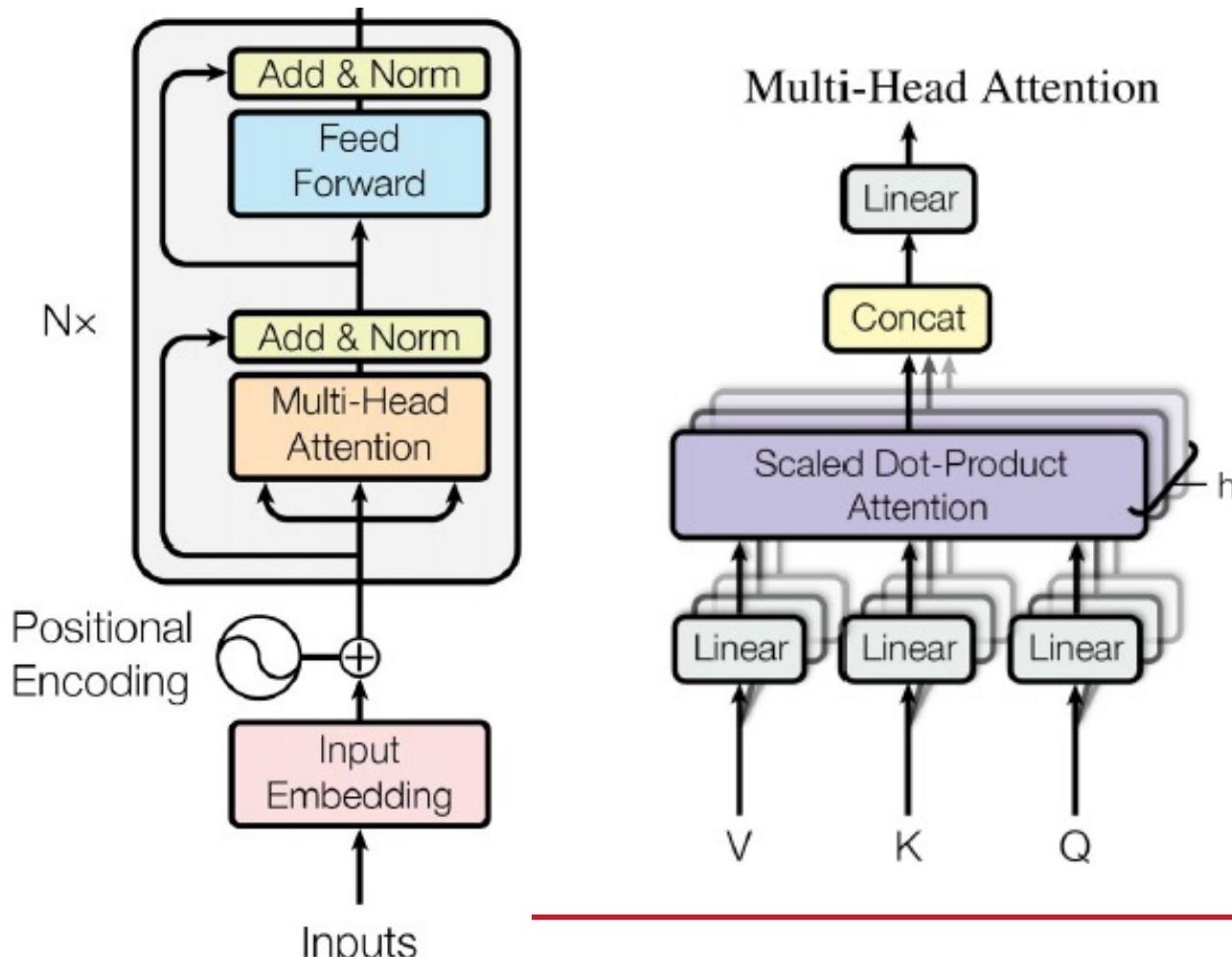
# Biểu diễn đầu vào

- Từ vựng 30,000 WordPiece.
- Mỗi token là hợp của 3 biểu diễn nhúng

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	$E_{\text{my}}$	$E_{\text{dog}}$	$E_{\text{is}}$	$E_{\text{cute}}$	$E_{[\text{SEP}]}$	$E_{\text{he}}$	$E_{\text{likes}}$	$E_{\text{play}}$	$E_{\#\text{ing}}$	$E_{[\text{SEP}]}$
Segment Embeddings	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_B$	$E_B$	$E_B$	$E_B$	$E_B$
Position Embeddings	$E_0$	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	$E_6$	$E_7$	$E_8$	$E_9$	$E_{10}$

# Kiến trúc mạng

- Transformer encoder

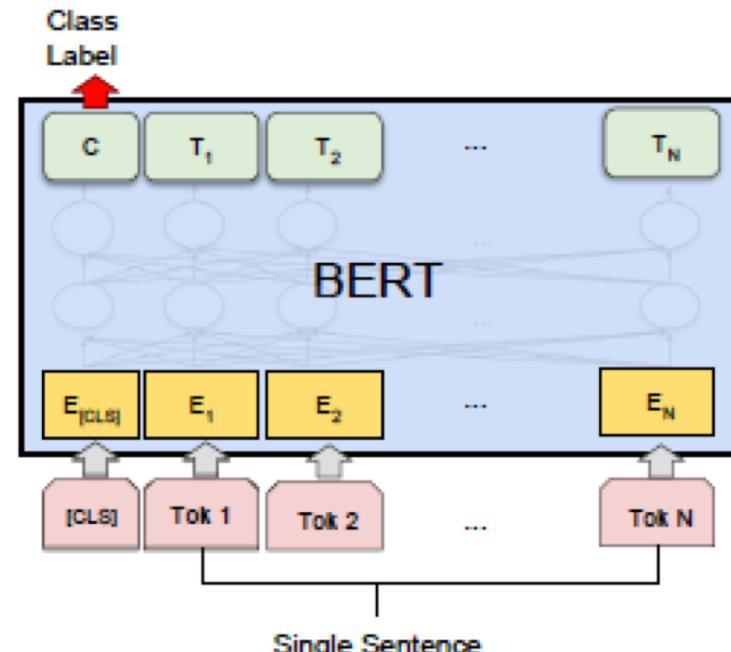


# Chi tiết mô hình

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences \* 128 length or 256 sequences \* 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: Adam, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head,
  - Total Parameters=110M
- BERT-Large: 24-layer, 1024-hidden, 16-head
  - Total Parameters=340M
- Trained on 4x4 or 8x8 TPU slice for 4 days

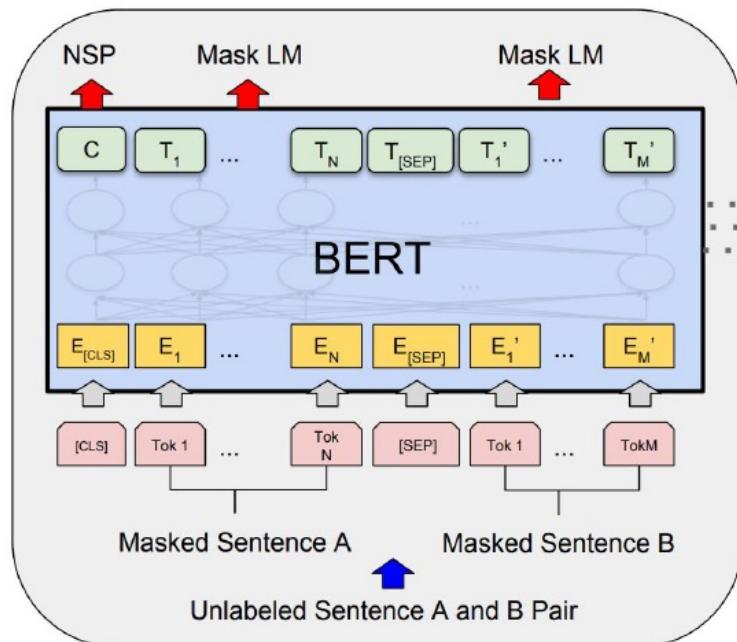
# Quy trình tinh chỉnh (Fine-tuning)

- Đối với nhiệm vụ phân loại chuỗi
- Lấy biểu diễn của chuỗi đầu vào bằng cách sử dụng trạng thái ẩn cuối cùng (trạng thái ẩn ở vị trí của mã thông báo đặc biệt [CLS])
- Chỉ cần thêm một lớp phân loại và sử dụng softmax để tính toán xác suất nhãn.

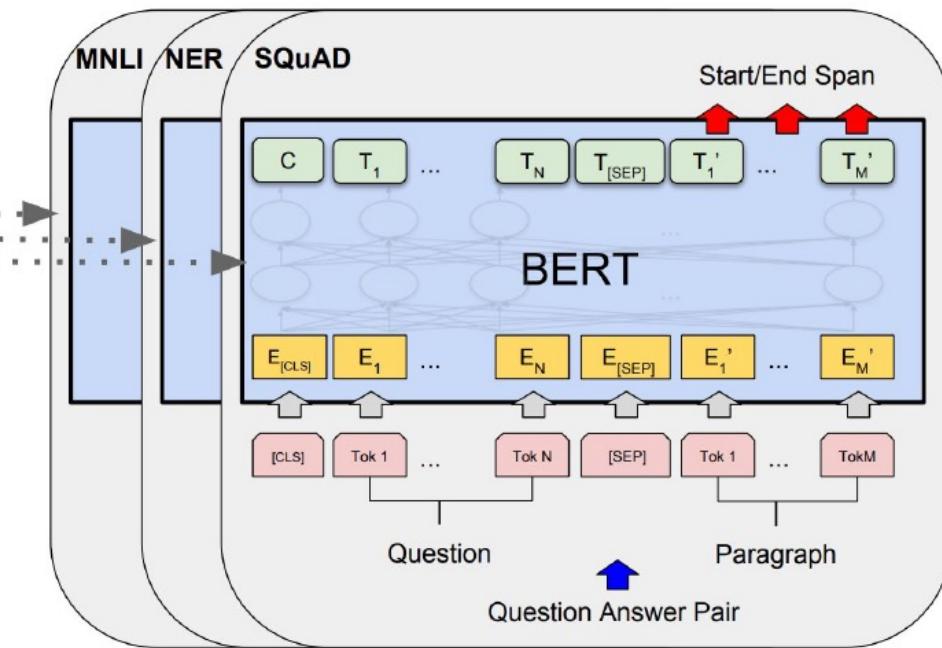


# Quy trình tinh chỉnh (Fine-tuning)

- Span-level task: SQuAD v1.1
- Biểu diễn câu hỏi và câu trả lời chung 1 chuỗi gồm 2 phân đoạn A (câu hỏi) và B (câu trả lời)
- Phán đoán Start/End Span



Pre-training



Fine-Tuning

# Thực nghiệm

- GLUE (General Language Understanding Evaluation)benchmark
  - Distribute canonical Train, Dev and Test splits
  - Labels for Test set are not provided
- Datasets in GLUE:
  - MNLI: Multi-Genre Natural Language Inference
  - QQP: Quora Question Pairs
  - QNLI: Question Natural Language Inference
  - SST-2: Stanford Sentiment Treebank
  - CoLA: The corpus of Linguistic Acceptability
  - STS-B: The Semantic Textual Similarity Benchmark
  - MRPC: Microsoft Research Paraphrase Corpus
  - RTE: Recognizing Textual Entailment
  - WNLI: Winograd NLI

# GLUE Results

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>

# Tài liệu tham khảo

Khoa cs224n Stanford:

<http://web.stanford.edu/class/cs224n/>



25  
YEARS ANNIVERSARY  
**SOICT**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**  
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

**Thank you  
for your  
attentions!**

